Outcome of UCL Workshop on the Architectural Aspects of Physics Analysis in Athena

Fredrik Åkesson, Kétévi Assmagan, Paolo Calafiura,
Kyle Cranmer, Samir Ferrag, Simon George, Jorgen Beck Hansen,
Roger Jones, Nikos Konstantinidis, Peter Loch, David Quarrie,
Srini Rajagopalan, David Rousseau, R.D. Schaffer, Peter Sherwood

Abstract

From April 5-7, 2004 a workshop was held at UCL to address the architectural aspects of physics analysis within athena. This document reviews the existing proposals; identifies underlying issues; and provides several new proposals. A timeline for these proposals and goals for the BNL software week are also presented.

1 Introduction and Goals

The purpose of the workshop was to address the architectural aspects of physics analysis within athena. The Reconstruction Task Force (RTF) addressed the architectural aspects of reconstruction, but their mandate stopped short of physics analysis. Let us quote the RTF proposal [1]:

> For the purposes of this report, the process of "analysis" has been defined in the following way: making analysis-specific cuts to existing objects, not making new objects. It is therefore necessary to have a processing step that goes from the Combined Reconstruction objects up to the input objects used by user analysis. This section describes this step, which is best described as "preparation for analysis" and the new objects and algorithms are part of the analysis data domain.

This quote represents only part of the starting point for the workshop. We review the other existing approaches in Section 2, and identify their common themes and fundamental differences.

The goals of this workshop [2] were to

- review existing approaches and identify their common themes and fundamental differences.
- consider use cases and identify requirements for analysis inside athena
- agree on an interface for common analysis tools
- provide proposals for the particle class
- clarify the short-, medium-, and long-term goals for those developing the analysis architecture

Because analysis and reconstruction are strongly coupled, some of the proposals in this document also apply to the reconstruction domain: *e.g.* the INavigable4Momentum proposal in Section 4. We also adopt a less restrictive view of "analysis": *e.g.* the IParticle proposal found in Section 5 makes it easy for a physicist to make new objects within their analysis. In Section 6 the "preparation for analysis" is fleshed out in terms of Particle Definition Algorithms.

In the remaining portion of this document we try to establish a loose time-line for how the analysis architecture will develop.

2 Review of Existing Approaches

At the A-team meeting on Monday, February 23rd 2004, the architectural model adopted by Artemis was discussed extensively. The main author of Artemis, P. Sherwood, was invited to that meeting. There were several questions about the new EDM (Event Data Model) interfaces, the adaptor classes, the wrappers for services such as StoreGate and the helper tools that were being developed by Artemis. Are these really needed or can we follow the Gaudi-ATHENA architectural model coupled with recent RTF recommendations to establish a framework for analysis? By evolving the existing EDM and framework functionalities, would it be possible to satisfy any requirements implemented by the Artemis approach? The workshop was originally organized to address these issues and recommend an architectural model for analysis.

In sub-sections 2.1 and 2.2 we will examine the relationship of the Artemis approach and the seeds of an analysis EDM sewn by the RTF. In the remainder of this section we will look at other approaches which do not address an analysis framework *per se*, but various aspects of it (e.g. navigation, tools, and the particle class).

2.1 Review of Relevant RTF Recommendations

As mentioned above, the RTF's mandate stopped short of the analysis domain; however, their report does envisage some aspects of analysis. In particular, the RTF specifies how the combined reconstruction domain and the analysis domain should interact (see Figure 1). In Section 6 we will discuss the analysis preparation algorithms in more detail. The RTF also addressed the need for a "common look and feel" for the user. They identified three approaches to provide this common look and feel:

- Inheritance
- Templates
- Adaptors/Wrappers

The pros and cons of these three approaches were investigated and each of the three approaches is employed at times in the atlas software.

The RTF proposed the I4Momentum interface for all 4-momentum like objects. I4Momentum provides a common look and feel through inheritance and several classes which implement 4-momentum manipulations for a specific "base" or representation.

The RTF also sketched the relationship of identified particles to a common base class (see Figure 2). Because I4Momentum does not provide particle id, vertex information, identification probability, navigation to constituents, or measurement errors it cannot be the base class. After reviewing the other existing approaches, a proposal for IParticle is presented in Section 5.

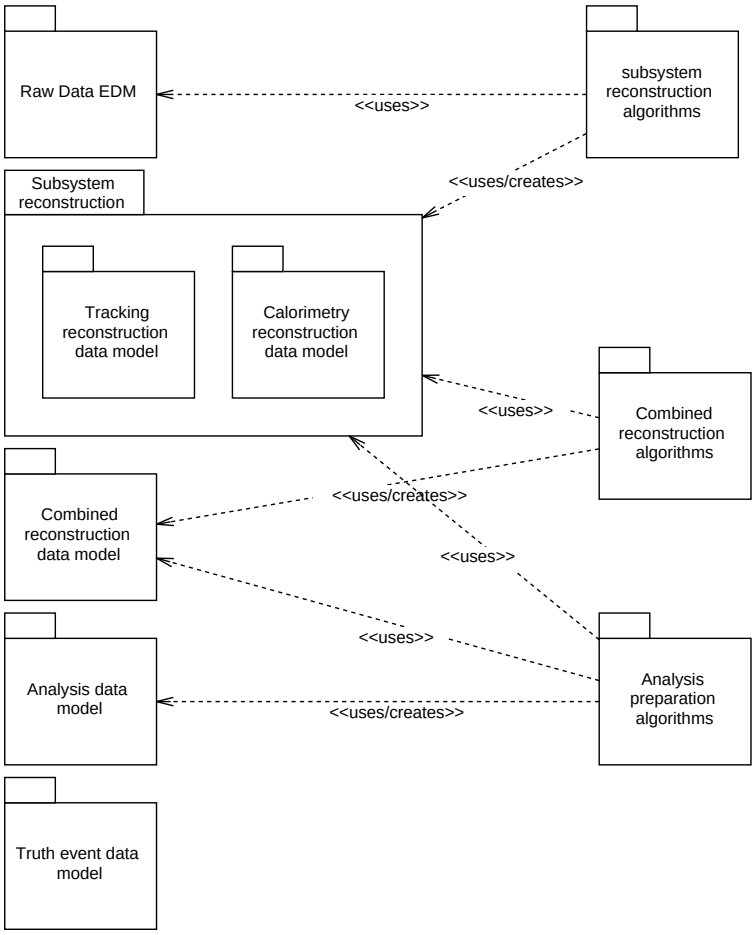


Figure 1: RTF Figure 2 showing the relationship between the combined reconstruction and the analysis domain.

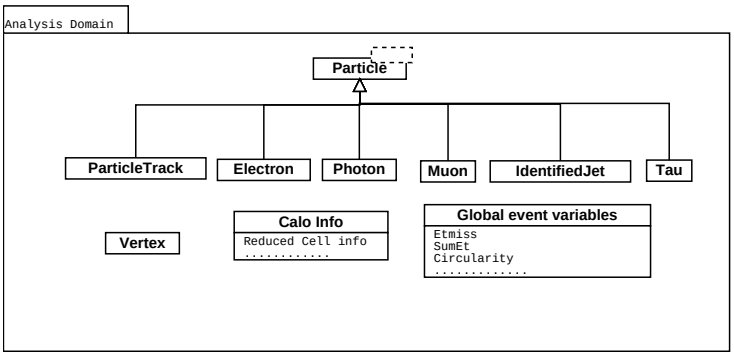


Figure 2: RTF Figure 23 showing several particle types inheriting from IParticle base class.

2.2 Artemis: Design and Functionality

Artemis is an analysis framework designed more than a year before the UCL meeting. Artemis provides:

- a standard interface to heterogeneous reconstruction classes (via the design pattern *Adaptor*)
- FromTDS class to automate redundant tasks
- some common analysis utilities (like sort by $p_\perp$)

Hierarchy

At the heart of the Artemis approach is the Artemis Hierarchy shown in Figure 3. At the top of the Hierarchy is `Artemis::IAO` - a base class which provides the `accept()` method for the visitor pattern and begin and end iterators for a list of associated IAOs. The associations provided by Artemis are discussed below.

Inheriting from IAO is IMomentum. IMomentum differs from I4Momentum because it has a different interface and because it inherits from IAO. There are currently no concrete implementations of IMomentum in Artemis; however, any concrete implementation would need to implement the visitor pattern and associations. On the other hand, I4Momentum does not provide associations or `accept()`. The presence of "unnecessary" methods in the momentum component of the interface was considered acceptable by the designers of Artemis because of the power of the visitor pattern and the ability to make arbitrary associations. The presence of these "unnecessary" methods was not considered acceptable for I4Momentum, where the clients are more numerous and varied. This "point of contention" was identified at the meeting and is the most fundamental architectural difference between the Artemis approach and the proposal for INavigable4Momentum and IParticle later in this document.

Adaptors & Simple Objects

When Artemis was first developed the various combined reconstruction classes lacked a common interface. Furthermore, Atlfast and the combined reconstruction lacked a common interface. In order to provide a common interface (a common look and feel) to the user Artemis adopted the design pattern Adaptor. For instance, all muons are dealt with via the Artemis::IParticle interface, although Atlfast, MuonBox, and Muid/Moore have different interfaces. The Adaptors are the concrete implementations of Artemis::IParticle, Artemis::IJet, etc. Most Adaptors have a pointer to the original object in StoreGate (which cannot be modified).

In order to manage memory, the adaptors use boost's "shared pointers". The current version of Artemis does not provide the user to access the pointer to the adaptee,

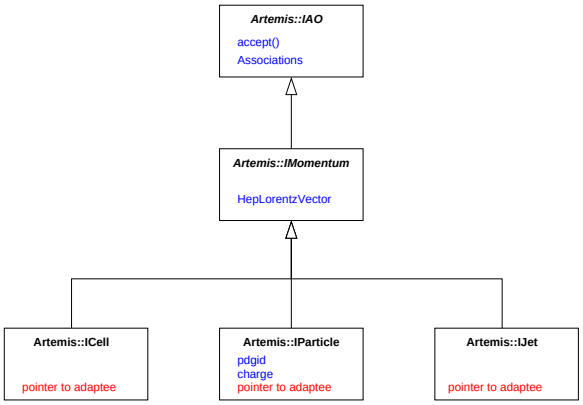


Figure 3: The Artemis Hierarchy

but that modification is being considered. Persistification issues are also being considered, but are not expected to be too difficult.

Because the user may need to create a new particle or modify the momentum or particle id, "simple objects" were created. Simple objects do not have a pointer to an adaptee in StoreGate, but instead have cached information.

Once the IParticle interface has been agreed upon and implemented and Atlfast can create the equivalent AOD objects, the need for adaptors will be diminished. However, the fact that Adaptors belong to the Artemis Hierarchy makes them different from the IParticle proposal below. The memory management features of Artemis and the need for an equivalent to simple objects is clear and discussed in Section 12.

FromTDS

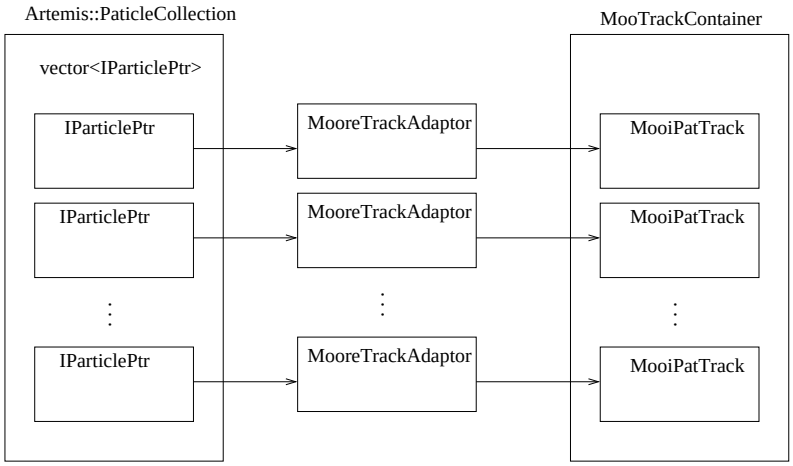
FromTDS is a class provided by Artemis which does not belong to the hierarchy in Figure 3. The use of FromTDS and the corresponding object structure is illustrated in Figure 5. The primary function of FromTDS is the templated `get()` method. The template parameter provides a number of typedefs necessary to specify the type of the source collection, the type of the source collection's elements, the adaptor to use, the adaptor builder to use, the type of the destination collection, and the type of the destination collection's elements (see Figure 4). Figure 6 shows the sequence diagram for `FromTDS->get()`.

The Functionality of `FromTDS->get()` can be emulated with standard StoreGate retrieve commands in conjunction with the IParticle base class and navigation. This requires

1 The collection is registered in StoreGate as it's concrete type and a symlink of that collection is made with type IParticle.

```
class MuidTrackCollection{
public:
  typedef MuidTrackContainer                              SrcCollection;
  typedef SrcCollection::value_type                       SrcPtr;
  typedef MuidTrackAdaptor                                Adaptor;
  typedef SimpleAdaptorBuilder2<MuidTrackCollection>      AdaptorBuilder;
  typedef ParticleCollection                              DestCollection;
  typedef DestCollection                                  DefaultType;
  typedef DestCollection::value_type                      DestType;
  static std::string description(){return "MuidTrackContainer";}
};
```

Figure 4: An example template parameter for FromTDS




```
ParticleCollection* myMooreMuons =          myFromTDS->get< MooreTrackCollection>()
ParticleCollection* myMuIdStandAloneMuons = myFromTDS->get< MuidTrackCollection>(m_muidStandAlonekey)
ParticleCollection* myMuIdComboMuons =      myFromTDS->get< MuidTrackCollection>(m_muidComboKey )
ParticleCollection* myMuonboxMuons =        myFromTDS->get< MuonboxTrackCollection>()
```

Figure 5: Below: Four examples of the usage of `FromTDS->get<T>()`. Above: UML diagram showing each `IParticlePtr` "has a" `MooreTrackAdaptor` which "has a" `MooiPatTrack`. The template parameter MooreTrackCollection is a collection of typedefs used by `FromTDS->get<T>()` (see text).

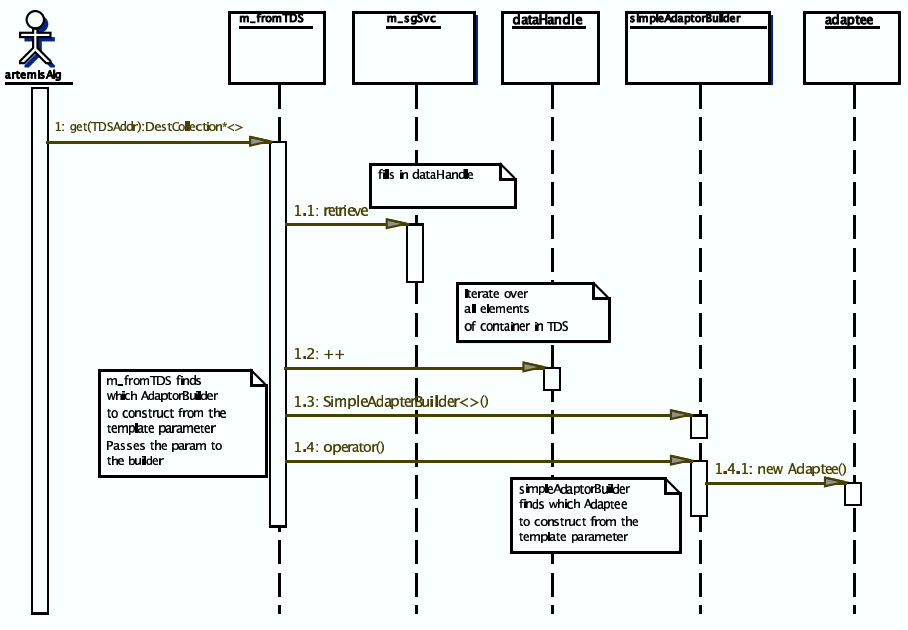


Figure 6: Sequence Diagram for FromTDS

2 The user retrieves the collection as IParticle

3 The user uses Navigation to recover the concrete type.

Tools

Artemis also provides a number of Tools (a.k.a. Helpers or Utilities) in the package ArtemisUtils. Some of these tools are purely kinematic tools, some make use of the visitor pattern, some make use of Associations, and some make use of Artemis::IParticle's extension to the IMomentum interface.

Early in the workshop it was pointed out that if IMomentum inherits from I4Momentum, then any momentum-like object in Artemis will be able to use a tool written on I4Momentum. It was also pointed out that if the tool returns an object of type I4Momentum, the object has "fallen out of the Artemis hierarchy". To return to the hierarchy, the user must dynamically cast back to IMomentum.

Example: A tool `I4Momentum* largerPt(const I4Momentum*, const I4Momentum*)` which returns the input argument with higher p_T. If the two input arguments are Artemis::IParticle, the returned object has fallen out of the hierarchy. An alternate

form of the tool could be
`bool largerPt(const I4Momentum*, const I4Momentum*,I4Momentum*)`, where the last argument is a return argument. In that case, downcasting could be avoided.

Modifications to Artemis were agreed upon so that Artemis will be able to use common tools written in terms of INavigable4Momentum. These modifications are discussed in Section 11. Even with these modifications, Artemis will be able to provide a super-set of tools which also take advantage of the visitor pattern and Associations.

Lastly, Artemis has begun the development of modular tools. For example, a tool has been created which performs n-nested loops over n-collections of Artemis objects, and for each entry in the sequence calls a generic "calculator". If the calculator produces a new collection, a "selector" can be called to make a subset of the new collection as the final return value. The "looper", "calculator", and "selector" are all modular, allowing them to be used individually or in the larger tool.

Associations

Artemis provides internal associations between arbitrary Artemis::IAO's. Artemis associations form a directed graph between IAO's (in contrast to the tree-structure of INavigable). Artemis cannot make associations to non-Artemis objects.

Early in the workshop a distinction was made between

- Navigation - Static structure reflecting composition of reconstruction objects.
- Associations - Dynamic associations not restricted to constituency

In particular, a muon might be associated with a jet, but not a constituent of a jet.

What is meant by "internal" associations is that the associations are stored in the objects themselves (i.e. an Artemis Jet might hold an association to Artemis muons internally). This is in contradistinction to "external" associations (i.e. a multimap of some sort). Internal associations have the disadvantage that once an object has been locked in StoreGate, the object cannot be changed – thus the associations cannot be changed.

The choice of Artemis' association strategy was largely influenced by the design choice for Atlfast associations. In Atlfast, associations are used primarily for composition tasks; thus, could be handled with INavigable. In the analysis domain, it is reasonable to suspect that users will want a more flexible structure. However, during the workshop no use-cases were found that could not be handled with a strategy more akin to INavigable.

It was also pointed out that it would be useful to have "labeled associations": a bookkeeping device for making different sets of associations for different reasons.

2.3 INavigable: Design and Functionality

The original motivation for object navigation came from jet reconstruction:

- Jet constituents are of generic type EnergyCluster; their concrete type is not exposed to the Jet itself.
- Clients need to retrieve objects of specific concrete type at any node of the relational tree behind a Jet: one needs a navigation system.
- Constituent objects in the tree can be composites themselves, thus navigation must be possible to any given level in the tree.
- Constituent objects can contribute their kinematics with a weight to the composite object, meaning weights must be retrievable and propagated correctly.

The design guideline for the navigation system consists of:

- All three object relations (composition, aggregation and inheritance) to be made navigable from any arbitrarily chosen node to any desired depth of the relational tree.
- General solution to honoring and/or forwarding a query for specific object types at any node of the relational tree with any relational parameter (not restricted to kinematic weights) suggests very straight forward implementation of composite objects and simple client interface for phrasing the query.
- Constituent collection inside a composite object means providing optimal collection type, support for persistency and query handling.
- All low-level methods to manage the constituent collection, i.e. all functionality to put, remove, find, and retrieve constituent objects, or manipulate their relational parameter. The EnergyCluster is a concrete implementation of the Navigable 4-vector:

The INavigable interface allows navigation to constituents (e.g. electron made of Cluster and Track, Cluster made of cells) through `Navigable<ConstituentContainerType>`. When implemented consistently in analysis and combined reconstruction domains, it will allow one to ask directly an electron for its CaloCell, for example. One needs the container type rather than the object type to allow for persistency. In cases of different constituent types (e.g. egamma has cluster and track), multiple inheritance is needed: `Navigable<TrackContainer>, Navigable<ClusterContainer>`. The positive aspect is the compilation time checking whereas a drawback is that it is not very flexible.

It is believed that this design could accommodate the "Identified Particles" domain, the final output of reconstruction, as well as the user analysis domain. The combined

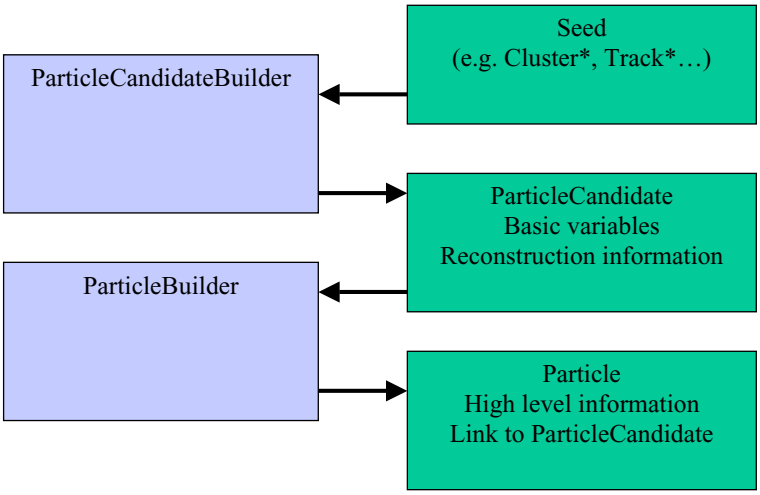


Figure 7: Algorithmic flow of information in PID.

reconstruction objects will probably not implement IParticle, but will probably implement INavigable4Momentum and certainly INavigable.

As a convention INavigable is for constituency (i.e. contributes to 4Momentum) and is not for associations. A similar structure could be used for Associations. This structure is appropriate for Composite Particles where the required constituency is honored.

2.4 PID: Design and Functionality

PID is a prototype to handle particle identification. A key feature of the PID design is the heterogeneous container ("Chameleon") capable of storing any data type indexed with a string key (retaining type information). The algorithmic or information flow is organized in a similar way as egammaRec to tauRec (see Figure 7).

The PID approach largely stresses that from a single particle candidate many particles can be created. The particles are unique (as specified by its quantum numbers). The PID philosophy is that a Particle should hold information about the choices that were made in creating it.

The PID approach attempts to template the structure of egammaRec, tauRec, etc... in a single package. This is the rôle of the ParticleCandidateBuilder. The ParticleBuilder, which takes ParticleCandidates and produces Particles, is similar to the Particle Definition Algorithms described in Section 6.

PID provides a particle with a common interface which can hold arbitrary recon-

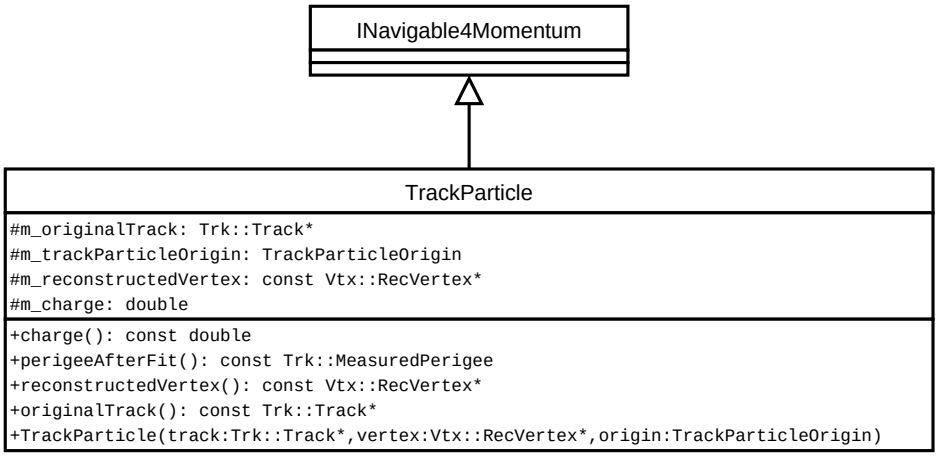


Figure 8: The Proposal for Track Particle

struction or analysis information. Persistency issues related to the Chameleon are under investigation and should be reported at the BNL software week. Expected modifications to the PID package are discussed in Section 12.

PID also provides some tools similar to the modular looper, calculator, selector discussed in Section 7.

2.5 Requirements of TrackParticle

As opposed to the other particle classes presented at the UCL meeting, the TrackParticle is not meant to be the candidate for a general purpose particle class. Instead, the TrackParticle is meant to be the interface for any algorithms which need tracks (e.g. vertexing, b-tagging, etc.). There is already a common track class (Trk::Track) which is used internally by the inner detector software and other subsystem reconstruction algorithms. The Trk::Track cannot be burdened with analysis domain requirements. Furthermore, Trk::Track uses a coordinate system that is natural for tracking but different from the coordinate system for physics analysis.

During the UCL workshop, it was agreed that TrackParticle would inherit from INavigable4Momentum, will have a pointer to a Trk::Track, and will have vertex information (see Figure 8).

2.6 BaBar-Inspired Particle

The BaBar inspired particle class is as single particle class largely tailored for the analysis seen at BaBar. The BaBar approach is somewhat similar to the PID approach in that from a single Particle Candidate or “Beta Candidate” many Particles can be

created. The BaBar approach uses shared pointers for memory management. The BaBar particle class also has most information one might need when doing an analysis (e.g. daughters, vertex information, truth information, pointer to track, pointer to a cluster, etc...).

The BaBar approach also provides a number of "combiner" classes which all implement the method `combine(list<Particle>)`. Combiners include adding two particles together to make a mother particle and vertexing tools.

The BaBar approach also allows one to change the information about the particle (e.g. the particle id, the mass, etc). Sometimes this change requires modification to the underlying Beta Candidate, sometimes not.

Most in the workshop agreed that there was a need to have a common interface for tools such as vertexing, that the navigation to daughters, and access to truth information were all valid requirements. However, there were objections to a "one-class-does-it-all" approach. Most felt that navigation to daughters and underlying detector information like tracks and clusters could be handled more elegantly and generally with INavigable. There was interest in the memory management approach and the fact that the BaBar particle holds pointers to the algorithms which created it.

3 Identification of Underlying Issues

One of the goals of the workshop was to identify the underlying issues between various design choices. We review the issues which were identified and discuss them briefly.

3.1 Hierarchy, Visitor, and Casting

The core issue which distinguishes the split inheritance (as seen in the design of INavigable4Momentum) and the hierarchy (as seen in the Artemis hierarchy) is the implementation of unnecessary methods. This was discussed in the **Hierarchy** portion of Section 2.2.

In the Artemis design, the IAO forces all sub-classes to implement the `accept()` method for the visitor pattern. The visitor pattern was discussed extensively at the workshop. The visitor pattern can be used to implement many tools, but is essentially a tool for type recovery. The key concerns with regard to the visitor pattern are the issues of scalability (scaling both in the number of visitors and the size of the hierarchy). A distinction was made between "cyclic" and "acyclic" visitors which illustrates the trade-off between maintenance and the (over-)use of casting and (slight) performance degradation [3, 4].

It should be pointed out that the INavigable design is a type of visitor pattern (where `fillToken()` is the equivalent of `accept()`). It should also be pointed out that

there is a dynamic cast when an INavigable object queries its constituents. It should also be pointed out that without the visitor pattern for type recovery, tools written with a return type of I4Momentum, may require an additional cast. The overall performance issues must still be addressed, but are expected to be acceptable.

3.2 Persistification Issues

Persistification is a omnipresent issue in Atlas software. The most common problem with persistification is that an object holds a pointer to another object. This pointer is not persistifiable, but can be replaced with an ElementLink or another equivalent. It was pointed out that at the end of an analysis, it is not necessary that all objects are persistifiable; however, it was generally agreed that an analysis should be persistifiable at the level of IParticle. Event Views and associations introduce even more persistification issues.

3.3 Trigger Issues

The main trigger-related issue is that of thread-safety. The HLT runs athena in a multi-threaded way. Any analysis code that is ported to the HLT must be thread-safe. Strings are known to be an issue, thus objects labeled by strings should be instead labeled with some sort of enumeration service.

4 INavigable4Momentum Proposal

The first agreement of the workshop was that the class currently called EnergyCluster (which inherits from INavigable and I4Momentum with no extensions) should be renamed INavigable4Momentum and that this class is the common interface for most objects in the combined reconstruction. The classes which will subclass from INavigable4Momentum include CaloCluster, CaloTower, egamma, TrackParticle, and the IParticle proposed below.

INavigable4Momentum is a pure interface, and subclasses must either implement navigation and momentum themselves or inherit from an implementation class. Typically there is a triple inheritance structure as seen in Figure 9.

4.1 Change to I4Momentum

The set methods of I4Momentum was discussed. In particular, should the set methods be public, private, or protected? It was decided that I4Momentum and IPxxxBase should not have public set methods, but that subclasses can have public set methods and modify the protected data members in IPxxxBase.

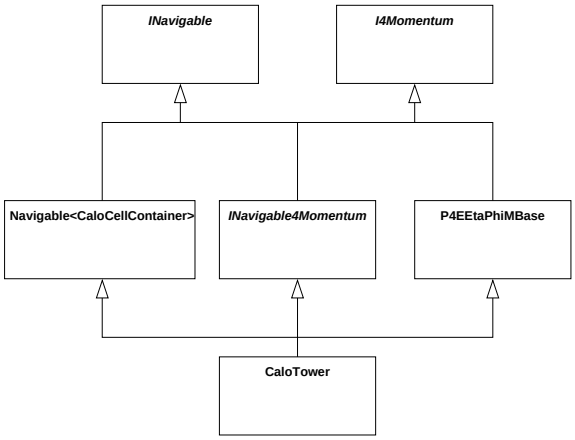


Figure 9: Triple Inheritance structure typical of classes inheriting from INavigable4Momentum.

4.2 IMeasured4Momentum

IMeasured4Momentum is a class which extends I4Momentum to include errors. This class needs to be thought through carefully; there is no proposal currently. The difficulty stems from the fact that for a track to be vertexed a 5-parameter error matrix is required and the naïve implementation would only have error on the four components of four-momentum.

5 IParticle Proposal

The second proposal from the workshop was that IParticle should inherit from INavigable4Momentum. Extensions to INavigable4Momentum are discussed below.

During the workshop four distinct use-cases for IParticle were considered:

- Particles from Reconstruction - For instance an identified electron which was produced from an egamma object. This electron should be able to navigate to the egamma object and should extend IParticle to include `EoverP()` and other egamma-related quantities. From the egamma object, one can navigate to a matched track or cluster.
- UserParticle - This is the equivalent of Artemis' simple objects. These are Particles that may have as constituents other IParticles (e.g. my particle has these daughters). UserParticles may also be able to have `setPDG(int)` methods which modify the particle id, mass, and recalculate the 4momentum as appropriate. Different choices in implementation might produce a number of similar classes.

- Particles with Forced Decays - Instead of Navigating to IParticles, a user may want to define a particle with a specified decay structure. For instance, a top decaying to a W and an b-tagged jet.
- Detector Level Particle - As emphasized by the BaBar inspired particle, a user may want to form an IParticle from just a track or cluster.

Each of these four use-cases can be satisfied with the IParticle Proposal (see Figure 10). The TrackParticle is another use case which was discussed with a specific implementation see in Figure 8.

Shortly after the UCL meeting, it was suggested that IParticle not necessarily include a particle hypothesis (*e.g.* a pdg code or a likelihood), and that a new class IdentifiedParticle be created for that purpose (see Figure 11). The consequences of this refinement will be investigated for the BNL workshop.

5.1 Particle Factory Proposal

Because analysis domain particles will be created by the user and possibly not registered with StoreGate (where memory management is implicit), some memory management tools are desirable. Artemis and the BaBar particles use shared pointers for memory management. The Artemis implementation uses a particle factory which is very user friendly. The detailed implementation of the particle factory will be presented at the BNL workshop. We need to address how difficult it will be for a user to define a class which sub-class from IParticle, the implementation of the factory for the new type, and the ability to persistify or register with StoreGate.

5.2 Extensions

The extensions of the INavigable4Momentum interface necessary for the IParticle interface were discussed briefly at the workshop. It was generally agreed that IParticle should have methods for accessing the particle's PDG id, the likelihood of the detector signature given the hypothesis (returned as a double), and a pointer to the vertex of the particle's creation. One should be careful about the ownership of the vertex – there should not be any problems if the vertices are always stored in StoreGate.

6 Particle Definition Algorithms

Particle Definition Algorithms (PDAs) are a key component of Analysis Preparation. PDAs take the output of combined reconstruction (which don't generally inherit from IParticle) and produce objects which belong to the "Identified Particle" domain. The

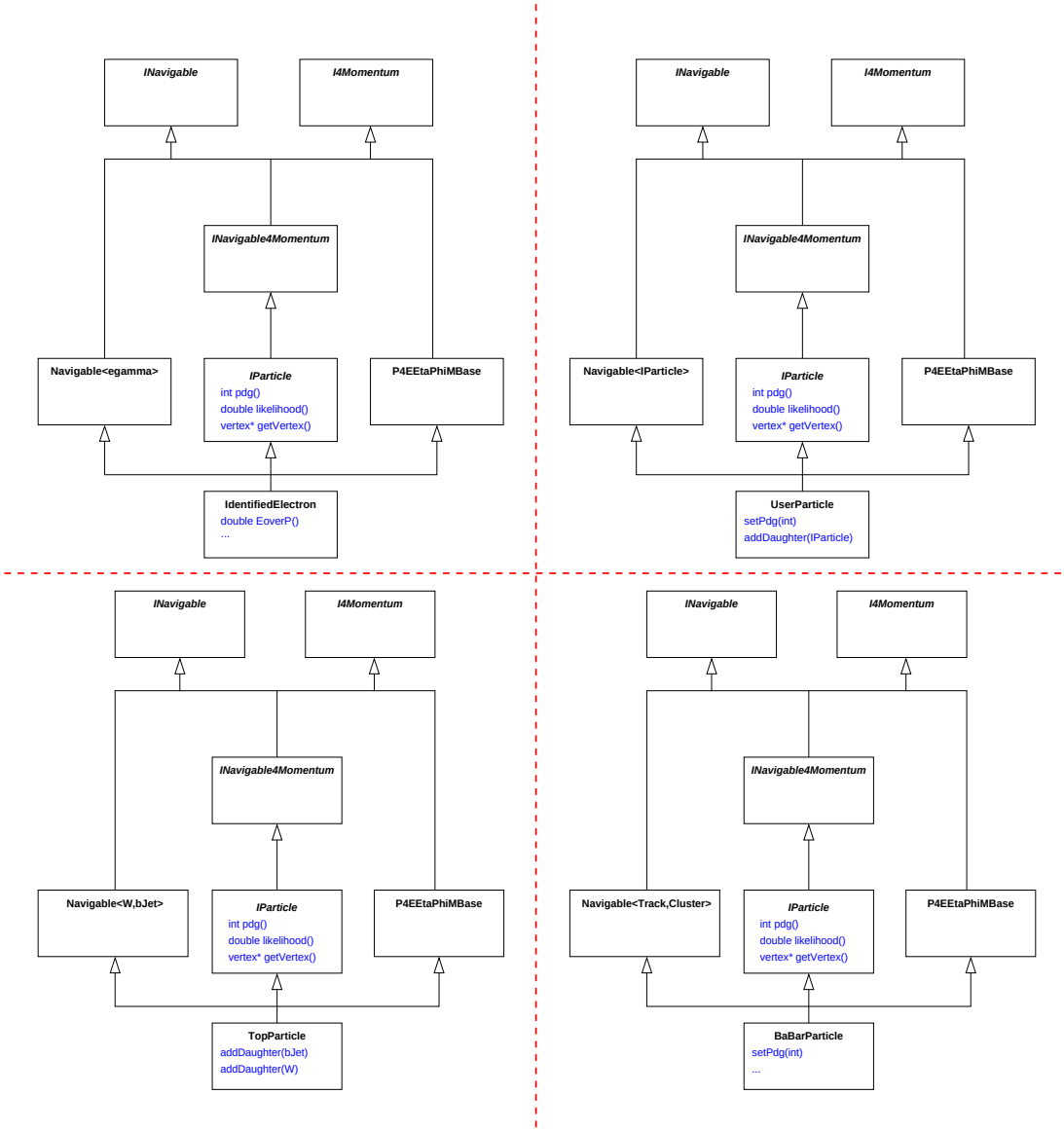


Figure 10: Four examples of concrete implementations of IParticle.

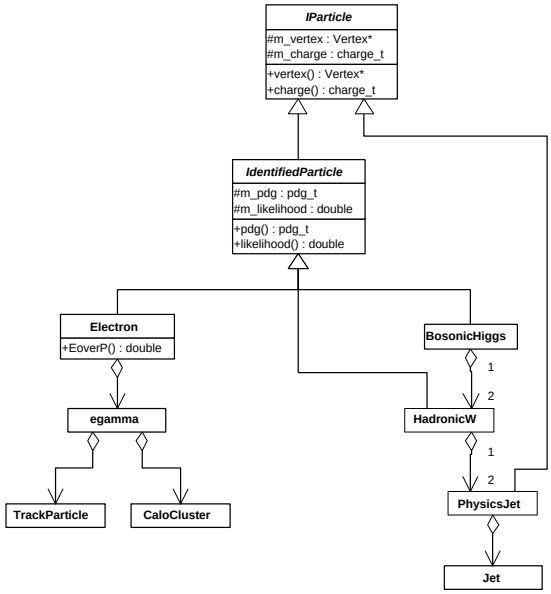


Figure 11: Possible Refinment of IParticle that puts particle hypothesis only into the sub-class IdentifiedParticle.

Identified Particle domain was not explicitly mentioned in the RTF proposal, but is seen as a well-defined stage after combined reconstruction and before analysis. Classes such as the IdentifiedElectron, IdentifiedMuon, etc... are expected to inherit and extend IParticle (see Figure 10).

In Figure 12 five example analysis workflows are presented. On the far left is an object from the combined reconstruction sits in the Transient Data Store (TDS). On the far right are five users' codes which implement the kinematic component of their analyses. The other boxes represent either algorithms which modify the default selection, or the output of those algorithms. The top-most workflow exemplifies an analysis which works directly with the reconstructed object using only default selection criteria. The second analysis exemplifies a user modifying the selection criteria in private code before proceeding with the kinematic component of his/her analysis. The third workflow shows an Artemis user either adapting the object with default selection criteria or modifying the selection criteria via an adaptor. The advantage of the top three workflows is that the user has complete control of his/her analysis; thus the possibility for this workflow will always exist. The disadvantage of the top three workflows are that modifications to the selection criteria cannot easily be used by other users; it promotes priviate code being developed making it harder for the physics groups to converge; and it discourages the adoption of standard particle definitions complicating systematic studies. These risk factors and general ease-of-use considerations suggest that there should be standard algorithms for producing Identified Particles from the output of combined reconstruction.

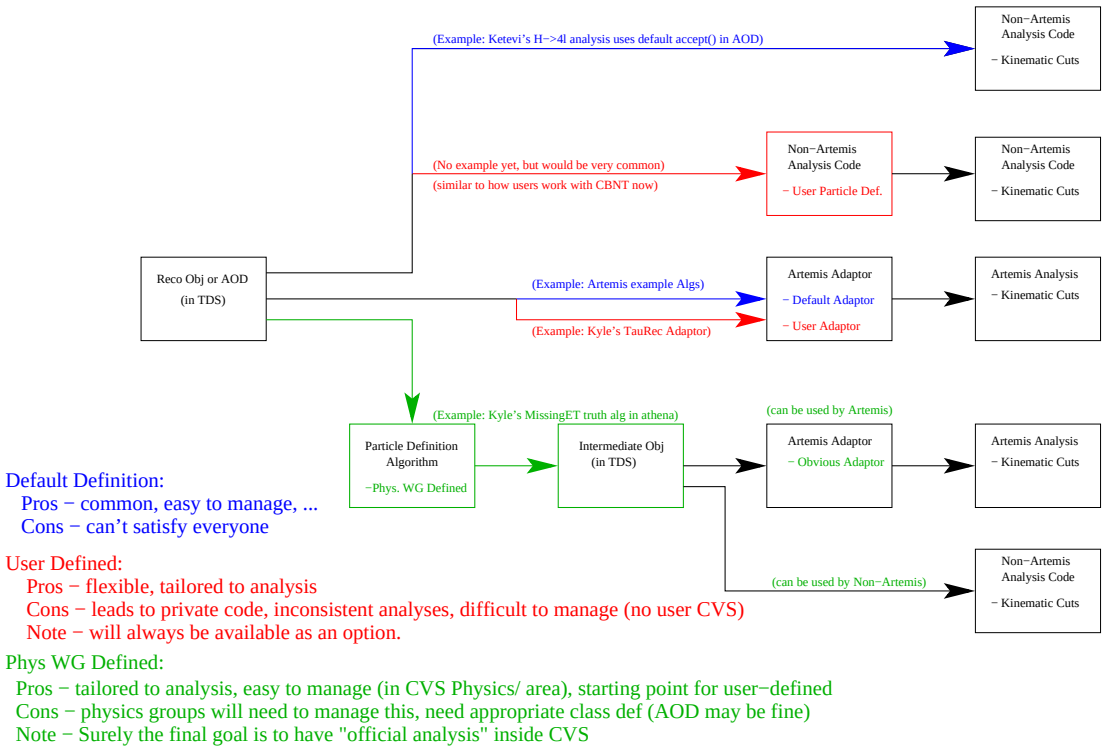


Figure 12: Five example analysis workflows from the output of combined reconstruction (left) to the kinematic component of analysis (right). Particle Definition Algorithms are seen as up-stream algorithms that convert combined reconstruction objects into the "Identified Particle" domain and implement additional selection, particle id, and kinematic calculations.

These Particle Definition Algorithms, are expected to be developed and maintained by physics groups. Undoubtedly they will first be developed by individual users and presented to physics groups for approval. Once approved, the physics groups can include them in the offline release where any user can have access to them via job options. This view of Particle Definition Algorithms suggests that they should be Gaudi Algorithms which register their output in StoreGate.

It is also natural to imagine that a user might want to use the Particle Definition Algorithm as a tool inside of his/her own analysis. This view of Particle Definition Algorithms suggests that they should be Gaudi AlgTools. During the March 3, 2004 Physics Analysis Tools meeting, it was decided that the algorithmic component of the tool should be implemented as a Gaudi AlgTool which produces only local, transient output and that a default Gaudi Algorithm should be provided to call the tool and register the output in StoreGate.

It should be pointed out that a single physics group might contribute several Particle Definition Algorithms - even for the same type of particle and same output type. For instance, the search for Higgs to four electrons might use a PDA that produces IdentifiedElectrons with the key "HiggsTo4eElectrons", while a SUSY analysis might use a different PDA that also produces IdentifiedElectrons with a different key.

7 Common Tools Proposal

The need for common tools is clear, and one of the goals of the UCL workshop was to decide on a common interface for those tools. It was decided to use INavigable4Momentum for most tools. For other tools that need pdg code, charge, or other information it seems natural to use IParticle as a common interface.

The INavigable4Momentum interface requires a change to the Artemis hierarchy if Artemis is to be compliant with these new tools. This modification is discussed in Section 12 and in illustrated in Figure **??**.

There was some discussion on modular tools like those being developed in Artemis and PID. Artemis has begun the development of a generic tool consisting of a modular "Looper", "Calculator", and "Selector" (see Section 2.2 **Tools**). Similarly, PID has a "fanout" tool, which may be incorporated into the analysis tools package.

With the common IParticle interface, it should be possible (even straight forward) to implement the "combine()" tools presented in conjunction with the BaBar-inspired particle.

8 UserAnalysis Package Proposal

One of the major challenges of a novice user is the creation of an athena package, the cmt configuration of a dual-use library, and the standard lines which setup a Gaudi Algorithm. The ArtemisUser subpackage is an example of a skeleton package that takes care of these standard things. It was proposed to make a similar UserAnalysis subpackage in the Physics package to take care of these basic steps for a novice user.

9 Event Views

The concept of an "Event View" was discussed during the UCL workshop in the context of a small group discussion. The goal of the discussion was not to arrive at an architectural proposal, but do define user requirements. Because "Event Views" is a new topic, we must first define it.

An "Event View" is a collection of physics objects which are coherent, exhaustive, and mutually exclusive. Event views are not unique; for each event a user may wish to consider the event with multiple different views. From this view, a user may wish to calculate several quantities (thrust, likelihood the event came from a given hypothesis, etc.) and associate it with the view (thus the collection of physics objects may include non-4momentum like entities). By coherent, it is meant that an analyst should not need to carry out additional checks or call additional tools to guarantee the consistency of the physics objects in a given view. For example, if a particle is assigned a certain PDG id, the relevant calibration constants should be used. Similarly, if vertexing is used by one view and not another, the list of vertices for the views should behave as expected. By exhaustive, it is meant that the sum of the energies of the physics objects should be roughly the total energy of the event and that the vector sum of the p_T of the physics objects should be roughly equal to $-p_T^{miss}$. By mutually exclusive, it is meant that the user is safe to consider any two objects as non-overlapping. For example, a jet should not also be listed as an electron; a track and a matching cluster should not be listed as separate objects, but put in some composite object. On the other hand, cells may be shared between two jets if their kinematic weights are taken into account. Clearly, what is meant by non-overlapping, or mutually exclusive, is not unique. Thus, alongside the algorithms that create event views are algorithms that check for overlap.

The overlap algorithms:

- must be flexible, configurable, and modular to aid construction of event views
- must allow the user to specify level to which overlap is evaluated (just check to the level of identified particles, or go down to the level of cells and tracks)
- support partial overlap (as in the case of cells in jets)

Event views are required by the user to be:

- easily specified (because they are not unique)
- driven by the analyst (via job options or run-time decisions)
- persistifiable
- navigable

10 Discussion on Associations

Associations were the most difficult issue for the UCL workshop. The difficulty stemmed partially from the fact that there were no example use-cases for associations that could not be satisfied through other means. It was agreed that what has been called "Navigation" should be restricted to constant types of associations; thus what is referred to (somewhat vaguely) as "Associations" are for non-constituent associations. The most obvious example would be a muon that is associated with a jet but not a constituent of the jet.

It was also agreed that the association strategy should support "labeled associations": sets of associations which can be distinguished by use of a key. The use case for labeled associations is as follows: one may wish to associate a jet with a muon for purposes of b-tagging and associate a different muon with the same jet as potential top-quark decay products; without a label to distinguish the associations this task is difficult.

In Artemis, each object in the Artemis Hierarchy has a list of associated IAOs (see Section 2.2 **Associations**). This structure forms a directed graph between IAO objects. Without some base-class like IAO, it is difficult to provide such general associations. Exacerbating the situation is the difficulty to decide on a base class fulfilling the role of the Artemis IAO, but spanning the Combined Reconstruction, Identified Particle, and Analysis domains.

Even if a base class was agreed upon (call it IAssoc), the implementation of the associations is not so straightforward. Because some associated objects may be locked in StoreGate, internal associations are seen to be problematic. External associations could be implemented with a multimap of some type, for instance:

```
multimap <A*, multimap<string, IAssoc>>
multimap <A*, multimap<Enum, IAssoc>>
```

There was no convergence on the association topic : it was agreed that the design should be modular enough so that codes and tools written on the INavigable4Momentum could be evolved easily when the association design is eventually chosen.

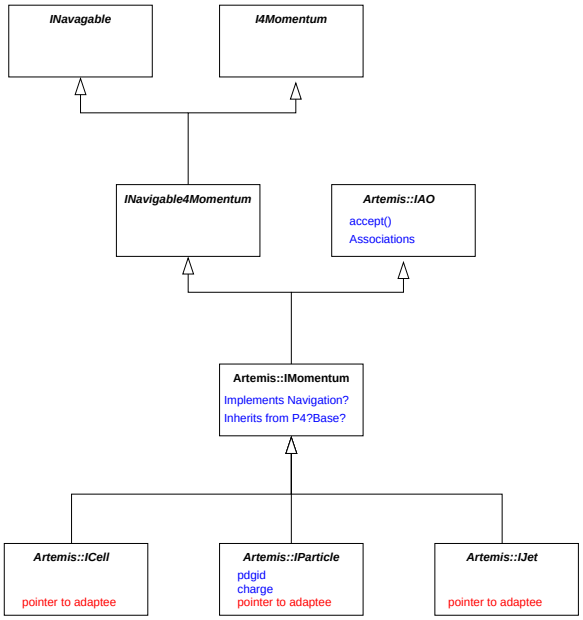


Figure 13: Modification to Artemis to be compliant with common tools written on INavigable4Momentum.

11 Modifications to Artemis

For Artemis to be compliant with the common tools written in terms of INavigable4Momentum, a modification to the Artemis hierarchy is expected. This modification is shown in Figure 13. The shortcoming of this modification is that any tool which returns an Artemis object in terms of INavigable4Momentum will need to be cast back into the Artemis hierarchy. This would only present problems if an analyst were to mix Artemis and non-Artemis objects in which case the cast might not succeed.

This modification to Artemis is expected to happen before the BNL workshop. It is not yet clear how Artemis will provide the implementations of INavigable and I4Momentum. I4Momentum can either be implemented by the Adaptors or by IMomentum. Also, it is possible that Adaptors could navigate to their adaptees. This functionality would be interesting, but would invite the analyst to mix Artemis and non-Artemis objects in a single analysis. The safest solution would be for IMomentum to inherit from NavigableTerminalNode.

12 Road Map for Analysis EDM

The analysis domain is still in its infancy. It is usueful to think of a plane spanned by two axes: complexity and ease-of-use. Artemis represents an analysis framework that

is of minimal complexity and very user-friendly. Analysis based on the output of the combined reconstruction (which was not designed for the analysis domain) represents an analysis framework that is high in complexity and not particularly user-friendly. The goal of the UCL workshop was to clarify the trajectory of both Artemis and the analysis EDM. Ideally, the analysis EDM will arrive at a point of maximal ease-of-use while still supporting very complex analysis requirements. Until the analysis EDM has made progress on a number of issues (outlined below), the rôle of Artemis is still quite clear. As the analysis EDM approaches its goal (referred to by some as "asymptopia"), the relationship between Artemis (which is also evolving) and the analysis EDM will need to be investigated further. The risk factor of having two analysis frameworks is for a single experiment is clear, but it's severity may be overestimated. The most dangerous aspects of two analysis frameworks is not the class a user stores their particle in, but the origin of those particles and the tools used to manipulate them. During the UCL workshop, modifications to Artemis were suggested that will ensure that Artemis clients have access to the same set of common tools. The common tools proposal is outlined in Section 7 and the modifications to Artemis are outlined in Section 11. The origin of the particles can also be made common by use of Particle Definition Algorithms (see Section 6).

It was generally appreciated that several pieces of functionality should be incorporated into the analysis EDM. Some of this functionality is inspired by Artemis, and some is new. Below we outline the most important functionality, comment on the plan of action, and indicate a rough timeline for completion.

- **Common Interface**: The UCL workshop was successful in agreeing that INavigable4Momentum will be the common interface for the relevant objects in the Combined Reconstruction domain. Similarly, it was agreed that IParticle will be the common interface for particle-like objects in the Identified Particle and Analysis domains. Already, a new package `Event/EventKernel` which holds I4Momentum.h, INavigable4Momentum.h, and dictionaries (no implementation) has been created. Also, a new package `Event/NavFourMom` has been created to hold INavigable4MomentumCollection.h and .cxx (and dictionary). TrackParticle is currently under development, as is the IParticle class. An example implementation of several IdentifiedParticles, the Particle Definition Algorithms, and the UserParticle are expected for the BNL workshop (see Section 12.1). The common interface for Associations was not found.

- **Common Tools**: It was decided that most kinematic tools will use the INavigable4Momentum interface, although some tools may use IParticle as the interface. Currently there are no common tools, though Artemis and PID have some for their respective interfaces. It is expected that by the BNL workshop, a package will be created and some common tools will be implemented. With the modifications to Artemis mentioned above, Artemis should be able to use these common tools.

- **Memory Management**: Just as Artemis and the BaBar particle use shared pointers for memory management, the analysis EDM should provide some memory management mechanism. Particle Factories and shared pointers may be implemented for the BNL workshop, where the implementation will be reviewed. The overall memory management strategy, the relationship to StoreGate, etc. will most likely continue to evolve for some time.

- **Generic Get, Select from TDS**: One piece of Artemis functionality to improve ease-of-use is the FromTDS::get() method. It was pointed out that similar functionality can be achieved with StoreGate's register, symlink, and retrieve commands together with a base class that is INavigable. The functionality is there in principle, but if it proves to be unacceptably complicated for the novice user, additional utilities may be needed.

- **Atlfast Compatibility**: The ability for a user to prototype code with Atlfast and then apply it to fully reconstructed events without rewriting (or even recompiling) is a key requirement for the Analysis EDM. Artemis provides this functionality with appropriate fast simulation adaptors. This functionality can be incorporated into the analysis EDM if Atlfast can fill the AOD in a sensible way. The AOD/ESD task force is addressing this issue. Until this functionality is available, Artemis will play a substantial rôle in analysis in Athena.

- **Associations**: Associations have been discussed extensively in this document. There was some progress made with respect to associations, but no general solution was found and no proposal is presented in this document. Artemis currently provides internal, unlabeled associations between Artemis::IAOs (and will probably provide labeled associations in the near future). The timeline for associations in the analysis EDM is not clear.

- **Event Views**: The concept of Event Views is very new, and one of the most high-level concepts proposed for the analysis EDM. Neither Artemis nor the analysis EDM currently provide Event Views, currently. Also, event view may store associations.

12.1 Short Term Plan

- K. Cranmer volunteered to start implementing UCL design of Figure 4, particularly the particle classes.

- S. Ferrag will come up with use cases and analysis examples based on the Babar inspired particle class.

- F. Akesson will proceed with the TrackParticle which is urgently needed as an input to combined reconstruction and for B-tagging.

- Chameleon persistency and adoption of INavigable4Momentum Association issues will be discussed further and resolved in the EDM and PAT meetings ; hopefully we will converge by the software workshop at BNL, May 23-28, 2004.

References

[1] RTF, Final Report of the Reconstruction Task Force, ATL-SOFT-2003-010.

[2] The UCL workshop agenda, http://agenda.cern.ch/fullAgenda.php?ida=a041436.

[3] Gamma, E. et. al., Design Patterns, Addison-Wesley, Reading, MA, 1994.

[4] Nordberg, M, Variations on the Visitor Pattern, http://www.cs.wustl.edu/ schmidt/PLoP-96/nordberg.ps.gz.